Software Engineering Institute

Combining Architecture-Centric Engineering with the Team Software Process

Robert L. Nord, James McHale, Felix Bachmann

December 2010

TECHNICAL REPORT
CMU/SEI-2010-TR-031
ESC-TR-2010-031

Research, Technology, and System Solutions (RTSS)
Software Engineering Process Management (SEPM)

http://www.sei.cmu.edu

Carnegie Mellon

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

Table of Contents

List of Figures

List of Tables

Acknowledgments

The authors would like to thank Bursatec for providing the opportunity to validate our ideas and the other members of the team for their technical contributions to the integration effort:

- Dr. Enrique Ibarra, Director General of Bursatec
- Luis Carballo, Manager of Software Engineering at Bursatec
- Greg Such, Carnegie Mellon® Software Engineering Institute (SEI) Business Manager
- John Klein, SEI Architecture-Centric Engineering (ACE) Initiative
- Gabriel Moreno, SEI ACE Initiative
- Andres Diaz-Pace, SEI ACE Initiative
- Jim Over, SEI Team Software Process (TSP) Initiative Manager

The authors also thank Nanette Brown, John Klein, Mark Klein, William Nichols, Jim Over, Linda Northrop, and Luis Carballo for their careful review of drafts of this report that helped to improve its content.

Abstract

This report contains a description of an architecture-centric life-cycle model that uses the Carnegie Mellon Software Engineering Institute's architecture-centric engineering (ACE) methods embedded in a Team Software Process (TSP) framework and of our experience in piloting the approach in an actual development effort. Combining ACE and TSP provides an iterative approach for delivering quality systems on time and within budget. TSP provides the infrastructure in estimation, planning, measurement, and project management. ACE provides the means for designing, evaluating, and implementing a system so that it will satisfy its business and quality goals. Bringing these approaches together results in something that is much more than the sum of the parts. The combined approach offers help to organizations to set an architecture/developer team in motion using mature, disciplined engineering practices that produce quality software quickly.

1 Introduction

In their keynote address on "Starting Right" at SEPG 2003 Europe, Watts Humphrey and Linda Northrop established the vision to combine the Team Software Process (TSP) with sound architecture-centric engineering (ACE) practices to accelerate projects and to produce better products.

The Carnegie Mellon® Software Engineering Institute (SEI) had the opportunity to realize this vision beginning in summer of 2009. At that time, the SEI began a project with Bursatec, the IT arm of La Bolsa Mexicana de Valores (the Mexican Stock Exchange), to replace its main online stock trading engine with one that would also incorporate trading of other financial instruments such as options and futures. The project had aggressive goals for performance and delivery, and as the face of Mexico's financial markets to the world, the new trading engine needed to function flawlessly.

The SEI answer to this challenge was to blend its ACE and TSP technologies. The TSP is a process for software teams. Its purpose is to build high-performance teams that plan, manage, and own their commitments; produce quality products at lower cost; and achieve their best performance. ACE is the discipline of using architecture as the focal point for performing ongoing analyses to gain increasing levels of confidence that systems will support their business goals. The combination of these two technologies creates a development environment in which teams can successfully build quality systems on time and within budget.

Combining ACE practices with TSP is now in the pilot stage. Similar approaches are also being piloted by others. Humberto Cervantes and his colleagues from Quarksoft and CIMAT have recently reported on their experiences in "Introducing Software Architecture Development Methods into a TSP-based Development Company" [Cervantes 2010]. The presentation describes an ongoing project whose aim is to introduce software architecture development methods inside Quarksoft, a leading Mexican software development company certified at CMMI® level 3.

The purpose of this report is to provide the description of an architecture-centric development process using TSP. This report is geared towards organizations that are starting a product development project that includes architecture and implementation activities, whether evolving an existing product or starting a new product.

This report begins by providing a summary of ACE methods and the role of TSP in Section 2. Background information is included to provide the reader with the necessary context; more detailed information about architecture practices [Bass 2003, Clements 2002, Clements 2003] and TSP [Humphrey 2002, Humphrey 2005, Humphrey 2006a, Humphrey 2006b, Nichols 2009] is described elsewhere. Section 3 provides the details of the combined approach and describes the architecture-related activities that are necessary during the architecture launch (executing and reviewing the plan for the architecture phase) and during the implementation launch. Section 4 explores the piloting of the approach in a project at Bursatec. Section 5 is a summary. Recommended training is described in the appendix.

® Carnegie Mellon and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

2 ACE and TSP

We begin by providing an overview of the ACE practices and the TSP iterative development model as context for understanding the combined approach.

2.1 Architecture-Centric Engineering Practices: A Brief Overview

ACE is the discipline of effectively using architecture(s) to guide system development.

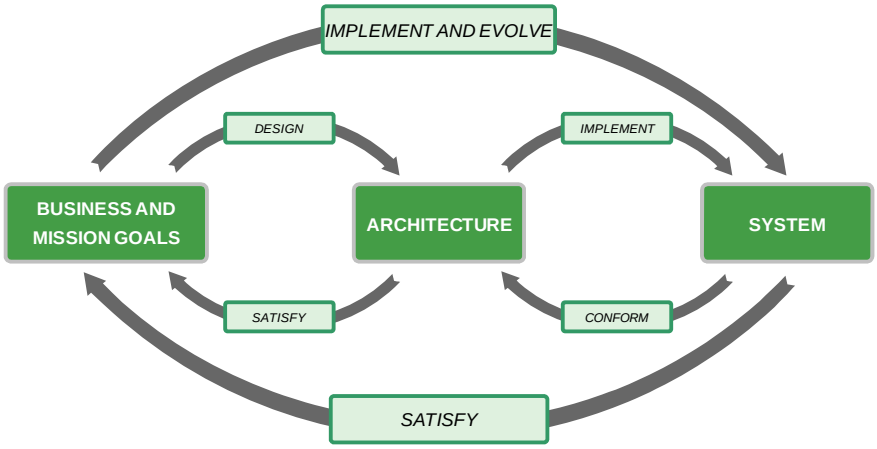

Figure 1: Architecture-Centric Engineering

Figure 1 shows the role architecture plays in ensuring that a system satisfies its business and mission goals during implementation and evolution. ACE analysis and design methods guide the creation of an architecture that successfully addresses the desired system qualities. The methods of interest for this report include:

- The Quality Attribute Workshop (QAW) [Barbacci 2003], to guide elicitation of quality attribute requirements from business and mission goals to the architecture. Quality attribute requirements are elicited from the system's stakeholders and are captured as quality attribute scenarios. Those scenarios are used as the driving force for architecture activities.
- The Attribute Driven Design (ADD) method [Wojcik 2006], to guide design from business and mission goals to the architecture. The ADD method provides a practical approach for developing an architecture to meet its quality attribute requirements. In architecture design, the defined quality attribute scenarios are used to iteratively decompose the system into an architecture that will fulfill the business goals.
- The View and Beyond (V&B) approach to documenting the architecture [Clements 2003], to guide the architecture team in producing architecture documentation that is useful to its stakeholders, easy to navigate, and practical to create.

- The Architecture Tradeoff Analysis Method® (ATAM®) [Clements 2002], to ensure the architecture satisfies the business and mission goals. The ATAM assesses the designed architecture with input from the system's stakeholders to uncover possible issues in the architecture early, before they create costly problems.
- Active Reviews for Intermediate Designs (ARID) [Clements 2002], to help hand off the architecture to the team implementing the system. The review focuses on whether the design is sufficient for the developers who will use it.

2.2 The Team Software Process: A Brief Overview

TSP is a development process enabling engineering teams to meet planned commitments, produce high-quality products, and deliver working software on time and within budget. TSP provides framework and a process structure for building and guiding self-directed teams.

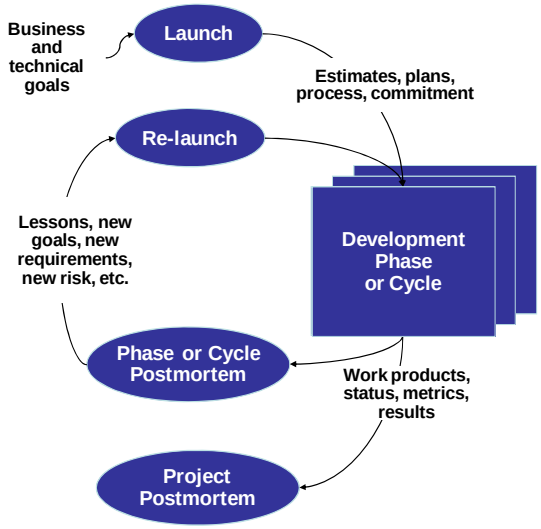

Figure 2: TSP Iterative Development

Figure 2 shows how TSP supports an iterative or cyclic development strategy. Products are developed over several cycles. Cycles may be organized into phases, according to the particular life-cycle development process in which TSP is used. TSP can be introduced at any phase or cycle. Each cycle starts with a launch or re-launch and ends with a postmortem. After each launch, TSP continues to provide guidance in managing the team through weekly meetings, checkpoints, and a postmortem.

The TSP coach guides the team through each launch, re-launch, and postmortem, and provides weekly coaching support during the cycle.

® Architecture Tradeoff Analysis Method and ATAM are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

3 TSP with ACE Practices

This section describes the approach to using the TSP with ACE practices to guide the architecture team in elaborating the architecture. We describe how to add architecture-related activities into a TSP development project that includes architecture design, detailed design, implementation, and testing.

TSP projects begin with a launch to build the team. Completing this launch requires some understanding of the number of teams that will be involved. Will one team do everything or will several teams do different tasks, such as an architecture team and a developer team. If the number of teams is not clear at the beginning of the launch, it will be clear after the launch is completed. The TSP launch described here is for a team whose responsibility is, among other things, to define or evolve the architecture of the product to be developed.

We describe in this section the architecture-related activities that are necessary during the launch performed by a team that is tasked with the architecture design. A TSP team manages itself according to the plan developed during the launch, so we describe how architecture influences the team members as they conduct their work. We also describe how the plan is reviewed and adjusted. Finally, we return to the concept of the TSP launch, only this time using it to transition from elaborating the architecture to constructing the implementation of the system.

3.1 Software Development Context

TSP can be used to manage all software development phases, from requirements elicitation to implementation and testing. Here we focus on the architecture phase with the launch of the team developing the architecture, the execution of the plan during that phase, and the handoff of the architecture to the developer team responsible for implementing the system.

The TSP launch for the architecture phase can be done when the following conditions are true:

- The team responsible for executing the architecture activities is defined. This includes the assignment of the team leader and the lead architect of the system. For small projects, the team leader might also assume the role of the lead architect.
- The major quality attribute scenarios are defined. This is usually done by executing a QAW.
- The major functions of the system are defined.
- The initial version of an architecture design strategy is defined.

The launch is led by the team coach. The team leader and coach understand TSP as well as the architecture methods. The team leader and coach have complementary roles to play in forming the team and in guiding the team members to carry out their work. The team leader is responsible for the project, leading the team to deliver a quality product on time and within budget. The team coach provides training, guidance, and feedback based on monitoring data and the process.

The requirements phase may still be underway, but enough work has been done to establish the vision of the future system, the business context, key system functions, and quality attribute requirements.

If different teams are responsible for the architecture design and implementation work, then there will be multiple handoffs of the architecture increments to the developer team. There also will be conformance checks of implementation products as architecture and implementation activities proceed in parallel.

3.2 Architecture Design Strategy

The goal of the TSP architecture phase launch is to plan the architecture activities in the context of supporting the organization's business goals and considering the existing time and budget constraints. Achieving that goal requires some initial understanding of the major components most likely to be included in the architecture, as well as an agreement on the major tasks needed to successfully design the architecture.

A default strategy that works in many cases and can be adjusted if no other strategy is available is to conduct the architecture design and implementation in iterations. The duration of an iteration depends on the complexity of the system to be developed. Six weeks per iteration can be used as a starting point. Table 1 shows the goals of the first five iterations of architecture and implementation activities, which may be performed by one or multiple teams.

Table 1: Activities and Goals for the First Architecture Iterations

Iteration	Architecture Activities and Goals	Implementation Activities and Goals
1	Create a candidate design using ADD with the major scenarios to structure the architecture. Conduct periodic ATAM-style peer reviews. At the end of the iteration, the problem areas in the architecture that require more detailed investigation and/or prototyping are identified.	Not applicable since architecture activities need one iteration of lead time in order to initiate implementation activities.
2	Use ADD to specify the well-understood areas of the architecture in sufficient detail that the architecture can be given to the developer team. Conduct periodic ATAM-style peer reviews. At the end of this iteration, the well-understood areas of the architecture are defined in sufficient detail.	Create prototypes to provide insight into the problem areas of the architecture. At the end of this iteration, data is available that helps design the problematic areas of the architecture.
3	Hand off the well-defined parts of the architecture for implementation to the developer team using ARID-style peer reviews. Use ADD to specify the remaining parts of the architecture, taking into account the results of the prototyping effort. Conduct periodic ATAM-style peer reviews. At the end of this iteration, all areas of the architecture are defined in sufficient detail and available for review.	Implement one function of the system in a partial skeleton system [Wojcik 2006] with the well-defined components. At the end of this iteration, the first system function can be shown to stakeholders.
4	Hand off the entire architecture for implementation by the developer team using ARID-style peer reviews. Conduct an architecture evaluation using ATAM. Refine the architecture to mitigate the risks uncovered by the ATAM. At the end of the iteration, version 1.0 of the architecture is available.	Implement a second function by implementing the whole skeleton system and the functionality required to provide the chosen system function. At the end of this iteration, a complete skeleton system with the additional running function can be shown to the stakeholders.

5	Refine the architecture and/or documentation to accommodate issues uncovered during the conformance review (see at right). At the end of this iteration, a stable architecture with sufficient documentation is available.	Conduct a conformance review with the architecture team members. Fix issues in the code uncovered by the conformance review and implement the next set of system functions. At the end of this iteration, the next functions can be shown to the stakeholders

If no major road blocks are encountered, the remaining iterations follow the schema as shown for Iteration 5.

3.3 Launching the Architecture Phase

During a TSP launch, the team reaches a common understanding of the work and the approach it will take and produces a detailed plan to guide its work.

The TSP launch is organized as a set of nine meetings over four days. The TSP launch process produces necessary planning artifacts (e.g., goals, roles, estimates, task plan, milestones, quality plan, and risk mitigation plan). The most important outcome is a committed team.

Table 2 shows an overview of the launch meetings and the architecture-related activities that are unique for launching a team at this phase in the life cycle. The team typically conducts detailed planning for the short term and overall planning for the entire lifespan of the project. The launch establishes a common team understanding of the project. During the launch, the role of Meeting 3 is enhanced since there is more information about the architecture design strategy to work with. Meetings 5 and 7 are streamlined since many of the issues they address are not known this early in the life of the project. Meeting 6 is streamlined since the investment made in Meeting 3 in understanding the architecture pays off when it comes time to use it in formulating the detailed next-phase plan.

Table 2: Overview of the TSP Launch for the Architecture Phase

Launch Meeting	Activities
1. Establish Product and Business Goals	The marketing or customer representative describes the desired product including the quality attribute characteristics.
2. Assign Roles and Define Team Goals	The team establishes goals that include building a system that meets the architecturally significant requirements. The team introduces new manager roles or modifies existing ones to incorporate architecture-related activities.
3. Produce Development Strategy	The team reviews the architecture design strategy of the desired product. The team establishes the project strategy; the results include explicit architecture deliverables. The team defines the development process; the process plan includes architecture design and evaluation activities and an architecture documentation strategy.
4. Build Overall and Next-Phase Plans	The team estimates the size of the architecture, taking into account quality attribute scenarios, views, models/prototypes, and iterations. The team includes peer reviews and tracking architectural risks in the project tasks.
5. Develop the Quality Plan	The team checks the quality plan against the quality-attribute-related team goals and the top-down plan to ensure consistency.
6. Build Detailed and Consolidated Plans	The team produces a near-term plan to determine and assign concrete architectural tasks and deliverables to team members and the team as a whole.
7. Conduct Risk Assessment	The team considers technical risks that inform the release plan and development strategies (e.g., prototypes, early development, and incremental versions).
8. Prepare Management Briefing	The team prepares the management briefing that includes quality attribute goals, architecture views, and risks and links them to the business goals and project schedule.
9. Hold Management Review	The team leader speaks to slides related to architecture: • Goals: design an architecture that meets the quality attribute goals • Deliverables: quality attribute scenarios, architecture documentation • Plan: based on ADD • Conclusions: role of architecture providing value

Meeting 1: Establish Product and Business Goals

The purpose of Meeting 1 is to review management goals and product objectives. Senior management and a marketing representative tell the team what they want the team to develop, when the product is needed, the resources available to the team, why the job is important, and how management will measure success.

Activities include the following:

- A senior management representative takes a more expansive view of the business goals to include all project stakeholders; the team listens for quality attribute goals of the development organization (e.g., strategic reuse, product lines, and buildability).
- A marketing or customer representative takes a more expansive view of the users' needs to include all product stakeholders and describes the quality attribute characteristics of the desired product when presenting the product objectives; the team listens for runtime and support quality attribute goals (e.g., performance, availability, and maintainability).

- The launch coach reviews the TSP team roles, including the role of the lead architect.

Functionality is often the primary focus when describing the characteristics of the product, and quality attributes are either implicit or taken for granted. Systems are frequently redesigned not because they are functionally deficient, but because they are difficult to maintain, or are too slow, or have been compromised by hackers. An explicit focus on quality attributes, such as maintainability, throughput, or security, during the launch sets in place the means to achieve them throughout design, implementation, and deployment of the product.

The team reviews the defined quality attribute scenarios to determine if they are still aligned with the business goals. If adjustments are made, the team defines some new scenarios and might drop others. At the end, the team has a set of scenarios that, if implemented successfully, will support the business goals. These scenarios form the basis for the planning activities in the next meetings.

TSP teams are self-directed with a leader whose role is to build, motivate, and maintain the team. On a project combining architecture practices and TSP, the lead architect works with the team leader. The lead architect's goals are to lead the team in producing the architecture, fully utilize all the team's skills and ideas in producing this design, and ensure that the architecture and its documentation are of high quality. The lead architect needs to be assigned prior to the launch, since this role is vital to forming the architecture design strategy for the project. The lead architect is a full-time job, more like a team leader role than a team manager role that requires a couple of hours over the course of the week. See Table 3 for details.[1]

Table 3: Role of the Lead Architect

Objective	When all team members consistently meet their roles' responsibilities, follow the defined process, and work to agreed goals and specification, the team will be most efficient and effective.
Goals	The lead architect's goals are to • Lead the team in producing the architecture • Fully utilize all the team's skills and ideas in producing this design • Ensure that the architecture and its documentation are of high quality
Role Characteristics	The characteristics most helpful to the lead architect are • Naturally assumes a technical leadership role • Is able to think in abstractions • Is able to identify the key technical issues and objectively make architecture decisions • Likes to design and build software-reliant systems • Has experience with architecture analysis and design • Has expertise in the domain and technology
Team Member Responsibilities	All team members, including the lead architect, are responsible for meeting their responsibilities as team members. • Meeting their team member commitments • Following a disciplined personal process • Planning, managing, and reporting on their personal work • Cooperating with the team and all team members to maintain an effective and productive working environment

[1] This table is modeled after the templates for the TSP team leader and design manager roles and responsibilities. In those templates, the rows labeled objective, goals, role characteristics, team member responsibilities, and principal lead activities are standard fields; others are included where they are specific to the role. Furthermore the description text for the objective and team member responsibilities is prefilled from the template and common for all roles.

Lead the Architecture	The lead architect maintains a focus on architectural issues throughout the project and leads the team to • Identify and resolve all architectural issues • Document and confirm architectural issue resolution • Focus on anticipating and addressing quality attribute issues • Produce, refine, and verify the product architecture • Use analyses, prototypes, or experiments as appropriate, to ensure that all the architecture issues and assumptions are identified, documented, and resolved • Ensure conformance of the implementation to the architecture
Manage Architecture Changes	The lead architect provides the team focus on anticipating likely change scenarios and designing for change as appropriate, balancing short-term needs with longer-term goals.
Establish and Manage Architecture Standards	The lead architect establishes the standards and procedures the team will use to produce the architecture design artifacts.
Principal Lead Architect Activities	The lead architect works with the team to perform their design tasks and resolve architectural issues. The lead architect reports at the weekly team meeting on the status of architecture standards and product design work.

Meeting 2: Assign Roles and Define Team Goals

The purpose of Meeting 2 is to set team goals and establish roles. The team reviews the management goals presented in Meeting 1 and derives a collection of measurable team goals. The team assigns the team management tasks among the team members.

Activities include the following:

- The team sets goals that include building a system that meets the architecturally significant requirements.
- The team reviews team manager roles that include architecture-related responsibilities.

The team starts with the business and product goals that management stated explicitly in Meeting 1. The team reviews and refines these goals, adding those implied by management. The team also adds team-specific goals. Management ultimately desires a product that produces some value and accomplishes that by managing the primary project factors of quality, function, cost, and schedule. If the product goals are not explicit, the team can look for those implied by the quest for quality and verify these when reporting back to management. These might be properties of the product itself, of developing the product, or of operating/managing the product. Goals need to be specified in enough detail that the team has confidence they can be accomplished in the plan.

Depending on its number of members and the scope of the project, the team may organize in a number of ways: a single team, a group of sub-teams, or multiple teams. An architecture design team is needed at a minimum. During design, it is common to spin off technology investigations to consider the selection of vendor-provided components or prototyping efforts to understand risks as they are uncovered.

In TSP, routine team management tasks are assigned to eight team member roles. Team members have TSP manager roles in addition to their team member responsibilities. The team divides the management roles, so that each member has at least one role responsibility. Every team goal needs to be allocated to a manager role to ensure that the goal is met. As a result, usual TSP roles may need to be tailored to ensure that the architecture-related goals are fulfilled. Activities related

to TSP manager roles should not take more than one to two hours per week. The person assigned a role is responsible for ensuring that the team addresses issues relevant to the role and reports on these during the weekly team meetings. Existing TSP roles have a different emphasis when including architecture-related activities (see Table 4).

Table 4: Role of the TSP Managers

TSP Manager Role	Architecture-Related Responsibilities
Customer Interface Manager	• Understands the business goals of the customer and the development organization and their priorities, understands the quality attribute requirements necessary to support those goals, and maintains traceability between the business goals and the quality attribute requirements • Manages quality attribute requirements issues and changes • Ensures all quality attribute requirements assumptions are verified
Design Manager	• Participates in technology investigations and on prototyping teams • Leads the design work and the design changes for the prototyping efforts
Implementation Manager	• Leads the implementation work for the prototyping efforts • Ensures that buildability issues are identified • Supports efforts for ensuring the implementation conforms to the architecture design
Test Manager	• Ensures that testing issues are considered in the architecture phase • Ensures that sufficient test cases are created to check the fulfillment of the quality attribute scenarios
Planning Manager	• Assists the team in using the appropriate architecture views (e.g., work assignment view, module view) to help in planning • Establishes the standard planning framework including architecture-related work products
Process Manager	• Leads the definition of the development process including architecture design and analysis activities • Supports transitioning architecture and TSP practices within the organization
Support Manager	• Establishes the development infrastructure (aligned with the implementation and install views of the architecture) • Oversees tools to support architecture design as part of the development support system
Quality Manager	• Ensures that the quality attribute requirements are well specified • Tracks architectural issues and risks in addition to defects

Meeting 3: Produce Development Strategy

The purpose of Meeting 3 is to produce the development strategy, development process, and support plan. The team members define the product that they will build and how to build it.

Activities include the following:

- The lead architect describes the architecture design strategy for the desired product.
- The team leader leads the team in establishing the project strategy with explicit architecture deliverables.
- The process manager leads the team in defining the development process with explicit architecture design activities.
- The support manager leads the team in reviewing development and process support tools and facilities and in defining usage conventions of the tools that may include architecture guidelines.

The lead architect leads the team in discussing or producing the conceptual architecture design with just enough detail to support project planning during the launch. While a detailed system architecture might not exist, it is typical that high-level system descriptions, context drawings, or other artifacts have been created that describe some of the system's technical details. In this case, rather than starting from scratch, the lead architect presents and builds on the system architecture descriptions as they stand with respect to these early documents.

Often, the existing context diagrams or high-level system diagrams describe a deployment view showing the system and how software is allocated in the computing environment or a conceptual layered view showing major software modules within the system and opportunities for reuse. Either of these may be used for planning purposes.

The lead architect leads the team in making a gross effort estimate for the architecture design. Architecture design activities fall into two broad categories that need to be estimated differently. The first category of activities is designing and analyzing an architecture to fulfill the quality attribute scenarios. The second category of activities is to specify the architectural elements (typically modules) in sufficient detail for the team members tasked with implementing the architecture. At the beginning of the architecture design phase, the first category of tasks usually will receive emphasis; towards the end of that phase, the second set of activities will be more prevalent.

Estimating architecture design tasks

Architecture design tasks can be estimated using quality attribute scenarios and the architecture components shown in the conceptual design. The quality attribute scenarios are classified by how difficult it will be to design a solution that will satisfy those scenarios in terms of high, medium, or low (H/M/L). The architecture components are classified by size (H/M/L). For every scenario, the team will determine which architecture components most likely need to be adjusted when they design for the scenario (see Table 5 for an example).

Table 5: Scenario / Component Mapping

	Component A (H)	Component B (L)	Component C (M)	Component D (L)
Scenario 1 (M)	X		X	
Scenario 2 (L)		X	X	X
Scenario 3 (H)	X		X	

Estimating module specification tasks

Architectural tasks focusing on designing for the specified quality attribute scenarios are not likely to produce architecture documentation that is sufficient for the developers. For example, neither concrete definition of responsibilities for each module nor specification of detailed interfaces for the modules is usually required to ensure that the architecture fulfills the quality attribute scenarios. But responsibilities and interfaces are very important to coordinating the effort of development teams to produce code that can be integrated and will run as expected.

Use cases help to discover module responsibilities and interfaces. However, producing good architecture documentation for developers does not require describing all use cases; it is sufficient to focus on the major ones. Major uses cases specify the main functions of the system. To identify those use cases, as a rule of thumb, check the following four categories:

- Operational—use cases that specify the purpose of the system (e.g., in a communication system, connecting and disconnecting; in a reporting system, creating a report)
- Administrative—use cases that specify major administration functions (e.g., administering settings for the application)
- Monitoring—use cases that specify monitoring function (e.g., current users online)
- Startup/shutdown—use cases that specific initialization and clean up functions

The team classifies those use cases according to complexity (H/M/L).

Establishing exit criteria for architectural tasks

Exit criteria that define when an architectural task is done promote a common understanding of the effort required to execute the task. For the two categories of architectural tasks—architecture design and module specification—two different exit criteria have to be established.

The exit criteria for the architecture design tasks need to define when a quality attribute scenario can be considered to be done. This can be achieved by peer reviews utilizing ATAM techniques. As soon as the architecture team thinks that a scenario is done, a peer review using at least one architect not involved in this project is conducted. The peer review results in a list of risks and possibly action items. The risks need to be mitigated, and the action items need to be executed. The design for a scenario is considered to be completed if all uncovered risks are mitigated and there are no open action items. A risk mitigation of "ignore this risk" is acceptable, if this designation is agreeable to the stakeholders.

The exit criteria for the module specification tasks need to define when the description of a module is good enough. This depends on the knowledge and skills of the developers charged with implementing the defined modules. This can be achieved using the ARID-style peer review, where the developers are tasked to sketch the solution for one or two use cases, using the provided architecture description. The result of the peer review is a number of suggestions for improvements. The module specification tasks are considered to be completed when all the suggestions that affect the documentation are resolved.

Remaining activities in Meeting 3

Estimating architecture design and module specification tasks, as well as exit criteria, are aspects of the architecture design strategy—one of the activities in Meeting 3. In the other activities

- The team leader leads the team in using the task classification to establish the development strategy, making the incremental architecture deliverables explicit.
- The team leader also leads the team in defining the work products. Associated documentation artifacts include quality attribute scenarios, use cases, architecture views, and supporting diagrams, descriptions, and analysis results. Manuals, training, and demos are among the other deliverables in which the architecture may be described for use by stakeholders to drive downstream life-cycle activities (e.g., testing, installation, monitoring, and operations).
- The process manager leads the team in elaborating the development process. The process plan includes a strategy and guidelines for architecture design, architecture documentation, and architecture evaluation.

Meeting 4: Build the Overall and Next-Phase Plans

The purpose of Meeting 4 is to produce the overall plan. The team builds a top-down plan for the entire job. It does this by estimating the size of the products to be produced, identifying the tasks needed to do the work, and estimating their effort.

Activities include the following:

- The lead architect leads the team in estimating the size of each work product. Sizing estimates for the architecture-related work products take into account quality attribute scenarios, architecture, analysis models/prototypes, and the iterative nature of design.
- The team leader leads the team in producing a task plan. The plan includes project tasks for periodic architecture peer reviews, tracking of architectural risks, and a final architecture evaluation. The lead architect has a role in carrying out project tasks throughout the overall development plan.

Tasks are defined for the duration of the project. They follow the team's process, include all products, and are detailed for the next phase. The time estimates for each task are based on size and productivity data or past experience.

Sizing and planning for the overall project can be done by revisiting the gross software sizing estimates of the principal product components that were inferred from the conceptual design in Meeting 3. Previously, the sizes of artifacts were estimated in terms of small to very large size, and low to high complexity. These estimates need to be translated into more precise numbers and assigned to a release cycle in the overall plan.

Estimating architecture design tasks

The use of historical data is recommended to estimate the effort for designing a quality attribute scenario according to the classification established during Meeting 3. If no historical data is available, the following rule of thumb can be used as a starting point. If a simple scenario (difficulty L) requires a small component (size L) to change, then this can probably be done within one day of effort. A complicated scenario (difficulty H) requiring a large component to be changed (size H) most likely requires one order of magnitude higher effort, that is 10 days. Other combinations fall in-between, as illustrated in Table 6.

Table 6: Effort Estimation Table (Days)

		Components		
		L	**M**	**H**
Scenarios	**H**	5	8	10
	M	3	5	8
	L	1	3	5

The purpose of these numbers is to provide a starting point. Historical data can support more accurate estimating. In any event, the real numbers will vary and depend on many factors such as the size of the project, the number of teams involved, whether development is distributed, and the skill level of the available staff.

Assigning effort numbers to the scenarios and their mapping onto components (shown in Table 5) would result in an effort estimation table like the one shown in Table 7.

Table 7: Example Scenario Effort Estimation Table (Days)

	Component A (H)	Component B (L)	Component C (M)	Component D (L)	Sum
Scenario 1 (M)	8	N/A	5	N/A	13
Scenario 2 (L)	N/A	1	3	1	5
Scenario 3 (H)	10	N/A	8	N/A	18

An architecture team usually works together; therefore the tasks cannot be executed in parallel by different team members. As a result, the effort numbers have to be multiplied by the number of team members working on those tasks.

Estimating module specification tasks

Historical data is also useful to estimate the effort for specifying modules with use cases according to the classification established during Meeting 3. If no historical data is available, the following effort estimations for use cases can be used:

- Easy use case—0.5 days
- Medium complex use case—1.5 days
- Complex use case—3 days

Typically two team members are assigned to a use case. Therefore the effort numbers have to be multiplied by two. Should historical data become available, these numbers can be adjusted accordingly.

Accounting for rework

When the architecture team designs the architecture for the second, third, and so on scenario, the design for the earlier scenarios probably will need to be adjusted. When following the default architecture design strategy (see Section 3.2), the following percentages of the overall effort for an iteration should be allocated to the rework of the existing architecture documentation, as a rule of thumb:

- 10% of the effort for Iteration 2 to adjust the scenarios from Iteration 1
- 20% effort for Iteration 3 to adjust the scenarios from Iterations 1 and 2

A recommendation is to plan at a gross level, and distribute the effort over the first two cycles and the remainder of the project life cycle.

Note that sizing the elements in this way does not explicitly account for infrastructure as separate elements; these are distributed among the other elements.

Sizing and planning for the work products of the near-term architecture phase of the project can be done more precisely by estimating the time needed to create the architecture documentation artifacts. These artifacts can be eventually translated into some size measures (e.g., pages of architecture documentation, number of artifacts in an architecture description), and effort can be allocated for producing them.

Documentation artifacts include views and supporting models. ADD suggests the design of the system will be represented using views from two or three categories (e.g., components and connectors, modules, or deployment). The Views and Beyond approach describes how each view is documented, with multiple diagrams to represent structure and behavior and text to describe the element catalog, rationale, traceability, and the like. To estimate the needed artifacts, the following rules of thumb can be used if historical data is not available:

- One quality attribute scenario used in Meeting 3 requires the creation or refinement of two structural diagrams, such as a component and connector view and a module view.
- One quality attribute scenario usually is refined into four more specific scenarios.
- Each specific scenario is described with three sequence diagrams.
- Each sequence diagram will require the definition or refinement of five architectural elements.
- Each architectural element has its own diagram showing its context.

Therefore, a quality attribute scenario from Meeting 3, on average, is described by two structural diagrams, twelve sequence diagrams, and five architectural element diagrams. The term *diagram* here is used to mean a visual representation of architecture elements and all necessary textual descriptions.

The design process may involve building analysis models and/or prototypes to understand and validate design concepts for important quality attribute requirements such as performance and availability. Once the models exist, architecture alternatives can be analyzed to determine the appropriate solution. The effort for building models and/or prototypes is not included in the estimation above.

The design process is iterative and incremental. After the initial handful of scenarios is addressed, a few more will be added to verify and extend the design. Accordingly, effort needs to be allocated for modifying the design decisions taken to address the initial scenarios and for adding a few more scenarios.

Finally, time needs to be factored into the plan for inspection every two weeks and a final evaluation when the architecture is stable.

When generating the overall plan, tasks continue for the architect throughout the development life cycle to maintain the architecture, guide the developers in using the architecture, ensure the implementation conforms to the architecture, and so on.

Meeting 5: Develop the Quality Plan

The purpose of Meeting 5 is to guide the team in producing the quality plan. The quality plan shows how the team will achieve its product quality goal. In TSP, software quality during product development is measured by counting defects and normalizing by the appropriate size measure.

Activities include the following:

- The team looks back at the quality attribute related team goals established in Meeting 1.
- The team checks the quality plan against the quality attribute goals and the top-down plan to ensure they are consistent and looks for needed adjustments.

There are two parts to producing the quality plan: (1) estimating where defects will be injected and (2) estimating where they will be removed. In the absence of historical data, TSP quality guidelines provide the standard planning factors shown in Table 8.

Table 8: TSP Quality Guidelines Standard Planning Factors

	Defects Injected / Hour	Defects Removed / Hour	Phase Yield
Requirements and Architecture	0.25	0.5 by inspection	70%
Detailed Design	0.75	1.5 by review 0.5 by inspection	70%
Code	2.00	4.0 by review	70%

Phase yield is the percentage of defects entering and injected in a phase that are removed in that phase.

Depending on the estimates, more- or less-aggressive inspections can be planned. The team can control certain activities (architecture design, inspection, tracking risks) and adjust them in accordance with quality goals.

Meeting 6: Build Detailed and Consolidated Plans

The purpose of Meeting 6 is to produce a balanced next-phase plan. The tasks for the next phase are allocated to the team members. The members build their own work plans using the estimation schema established in Meeting 4. The team balances the workload so that everyone completes their next-phase tasks at approximately the same time. The team merges the individual work plans to form a consolidated team plan.

Activities include the following:

- The team allocates the tasks for the next phase to individual team members using the following guideline: during the architecture phase, the team plans to work together as a group through the early stages of requirements analysis, architecture design, and review. This plan is used by the team to guide and track its work during the upcoming project phase.

The architecture team works as a group during the early iterations of design, since it analyzes the global factors that influence the architecture and makes decisions that affect the structure of the product solution. Once enough of the structure is realized, there will be more opportunities to divide the work of pursuing the decomposition of identified subsystems, mitigating risks by building prototypes, completing the documentation of the design concept, and so on.

Meeting 7: Conduct Risk Assessment

The purpose of Meeting 7 is to conduct a project risk assessment. The team identifies and assesses the risks to its project plan. Risks are analyzed to determine impact and likelihood and assigned an owner for investigation and tracking. Also, a mitigation strategy is noted for high and medium priority risks.

Activities include the following:

- The team considers technical risks and their impact on the business goals of the project.
- For the higher priority risks, the team identifies mitigation actions that impact the development strategies (e.g., prototypes, early development, and incremental versions). The development strategies will inform the release plan.

Some architecture-related risks can be identified at this point and are relevant to the project risks that are reported during this meeting. These architectural risks may pertain to organizational awareness of the activities needed to support architecture, architecture support for the achievement of qualities, uncertainty over requirements or the scope of the product [Bass 2006]. Making risks explicit enables them to be discussed, so that they can be properly managed.

Meeting 8: Prepare Management Briefing & Meeting 9: Hold Management Review

The purpose of Meetings 8 and 9 is to prepare for and conduct the final launch management meeting. The team prepares and delivers a presentation of the project plan to management. Management probes the team's plan to assess the quality of the team's work and decides if the plan is acceptable.

One activity enhances Meeting 8 and Meeting 9 of the TSP launch: The team considers additions to existing templates to report on quality attribute goals that support management's overall goals, architecture deliverables that support monitoring of progress, and any identified architecture-related risks and their impact on business goals.

Architecture topics of interest to management include the following:

- Goals—design an architecture that meets the quality attribute goals
- Deliverables—quality attribute scenarios, architecture design and documentation, analysis models and prototypes
- Plan—based on using ADD and building models/prototypes, on peer reviews, and on evaluation using the ATAM
- Conclusions—role of the architecture in contributing value to the project

Documenting architecture views is one of the activities yet to be performed by the architecture team; however, some documentation is likely to be done at this point, such as a top-level context diagram showing the relationship of the system of interest to its environment, a conceptual layered diagram showing major elements of the system from a marketing point of view and opportunities for reuse (some call this a "marketecture"), or a view showing the major work products and their assignment to teams.

The ultimate purpose of including information about the architecture in the presentation is to increase management confidence that the business goals can be achieved given the time and resources allotted in the project schedule. The architecture artifacts serve two purposes, as a design artifact that can be analyzed to demonstrate support for the business goals and as a blueprint that guides development activities within the constraints of the project schedule.

3.4 Executing the Plan

The team manages itself according to the plan developed during the launch. ACE techniques—the ADD method, Views and Beyond approach, and ATAM—supplement and strengthen the architecture phase.

During a TSP cycle, there are shorter time periods of work. According to TSP guidelines, the team meets for one hour weekly to review the past week and plan for the week to come. In addition to the weekly planning meeting, a regular team meeting is useful for architecture design dis-

cussions. The team reviews the evolving design concept, using peer review based on the architecture analysis activity of the ATAM, to analyze the current architecture design with respect to the quality attribute scenarios that are the focus of concern [Edmondson 2007, Forstrom 2008]. One team member explains the architecture and the others on the team play the role of the evaluation team, asking questions and probing for risks. It is always a good idea to include an ATAM-trained architect, not involved in this project, in the peer review. The team then looks ahead to plan for the next interval of design work, working through the issue list and revisiting the plan. They decide on the problem to tackle next, sketching some design alternatives, all the while making sure that progress is being made.

To guide its work, the team uses the ADD, a decomposition method based on transforming quality attribute scenarios into an appropriate design (Figure 3). The architecture team scrutinizes the requirements (including constraints, functional requirements, and quality attributes) to identify the candidate architectural drivers. Typically there are a handful of drivers; these are the quality attributes scenarios that reflect the highest priority business goals and that have the most impact on the decomposition of the architecture. The number of iterations (influencing the depth of the decomposition) and the order of in which the decomposition tree is developed will vary based on the business context, domain knowledge, technology, and so on.

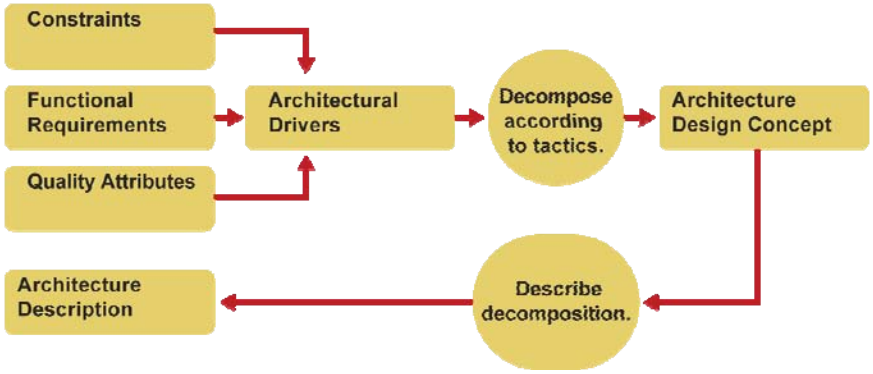

Figure 3: ADD Conceptual Flow of a Single Iteration

The early steps of ADD are driven by the quality attribute scenarios that influence the architecture design concept. The architecture design concept consists of the major types of elements that will appear in the architecture and the types of relationships between them. Often these are in the form of patterns or architecture styles.

Quality attribute requirements are an important input to architecture design. Quality attribute requirements are represented as six-part quality attribute scenarios so they are clear and unambiguous. The six parts are as follows: what condition arrives at the system (stimulus), who generates it (source), what it stimulates (artifact), what is going on at the time (environment), and the system's reaction to the stimulus (response) expressed in a measurable way (response measure).

Figure 4 shows a diagrammatic representation of the six parts of the following availability scenario: An unanticipated external message is received by a process during normal operation. The process informs the operator of the receipt of the message and the system continues to operate with no downtime [Bass 2003].

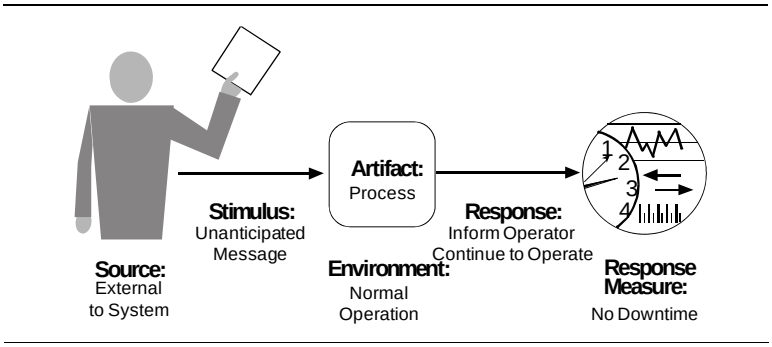

Figure 4: Sample Availability Scenario

During later steps, the focus turns to more fully documenting the design. When a scenario is achieved, it is documented with views for that scenario along with the rationale. Justification may include the results of any analysis models or prototypes. The goal is to add detail to the patterns established in the previous iteration, complete the documentation package, and validate the design with the remaining requirements. The architecture of the product must be documented in a way that is helpful to the development organization. The Views and Beyond approach to documenting software architecture provides examples of architecture views and design templates to guide the work. It can be used to define what needs to be documented in a format that is helpful to the stakeholders of the system, including developers, testers, project manager, and steering committee.

Using the Views and Beyond approach, documenting a software architecture is a matter of documenting the relevant views and then adding information that applies to more than one view. Figure 5 shows that to document a view, use a standard organization consisting of six sections [Clements 2003].

1. The primary presentation shows the elements and relationships among them that populate the portion of the view shown.
2. The element catalog details those elements depicted in the primary presentation.
3. A context diagram shows how the part of the system represented in the view relates to its environment.
4. A variability guide shows how to exercise any variation points that are a part of the architecture.
5. Architecture background explains why the design reflected in the views came to be.
6. Other information will vary according to the standard practices of the organization or the needs of the project.

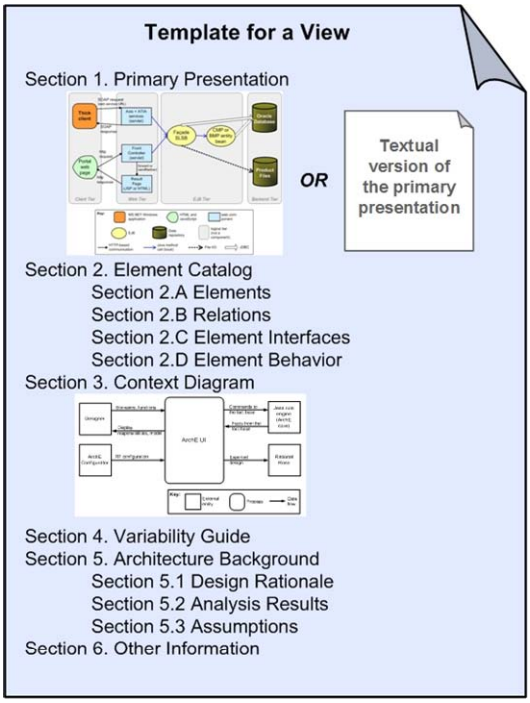

Figure 5: A Template for Documenting a View

The results of each iteration of the ADD method can be captured in the view template. At the beginning of an iteration, a blank view template is used that includes pointers to the view's chronological predecessor, parents, and siblings, if any. In the Design Rationale section, the requirements are documented with an explanation for why this element was chosen to be designed when it was. Also, in that section, the drivers for this element are documented with the motivation for the selection of the pattern/tactics. The instantiated pattern/tactics are documented in the primary presentation and the element catalog. The elements' interfaces are sketched in the interface section of the element catalog.

As soon as the architecture begins to stabilize, it is reviewed with input from the broad stakeholder community to uncover possible issues in the architecture before they create costly problems. An evaluation of the architecture by an external team is an important capstone event in the life of the project. The previous peer review evaluations were done with internal team members. The complete evaluation at the end of the architecture phase is designed to give the stakeholder community confidence that their concerns are understood by the architecture team and that the architecture addresses them. Other benefits include additional identified risks, clarified quality attribute requirements, and increased communication among the stakeholders.

The ATAM relies on enlisting stakeholders to help analyze the architecture. The ATAM evaluation team guides the stakeholders in elaborating the business drivers and quality attributes of the system in the form of scenarios. The scenarios are used to analyze the architecture to understand tradeoffs, sensitivity points, non-risks, and risks. Risks are distilled into risk themes, so that their impact on the architecture and the business drivers can be understood.

The evaluation portion of the ATAM is conducted in two phases at the customer site. The phases are distinguished according to the class of participants needed and the way quality attribute scena-

rios are elicited. Phase 1, the initial evaluation, involves a small group of predominantly technically-oriented stakeholders. This phase is architecture centric, focused on eliciting detailed architecture information and top-down analysis. Phase 2, the complete evaluation, involves a larger group of stakeholders. It is stakeholder centric, focused on eliciting points of view of diverse stakeholders and verifying the Phase 1 results.

The outcome of Phase 1 has been achieved during the peer reviews. Given this context, the architecture evaluation can be streamlined and focus on ATAM Phase 2 activities.

When the documentation package is sufficiently complete, it can be used by the developer team.

3.5 Reviewing the Plan

TSP concludes each phase or cycle with a postmortem to assess progress against the plan; discern lessons learned; and gather new business goals, requirements, and risks that are input into the launch of the next phase. This cycle post-mortem is a comprehensive study of team performance and data and includes recommendations or proposals for changes that can assist team performance improvement.

Following the postmortem, the coach guides the team through the re-launch for the next cycle. TSP re-launches have two fewer formal meetings than do launches. Meetings 8 and 9 are not included in the TSP re-launch, although the team leader typically summarizes re-launch results to management in private. Primarily because those meetings are not included, re-launches are nominally scheduled for three days rather than the four allotted for initial project launches. Another reason for the shorter re-launch schedule is that Meeting 1 tends to require less time than in a launch since it is mainly a status report by the team leader.

A TSP re-launch can be as straightforward as doing detailed planning for the next phase of a project that already has a well-structured overall plan developed during a previous launch. It can also be as complicated as throwing out the previous plan based on current, presumably superior, understanding of the project's requirements, constraints, and other realities. Most often, the reality is somewhere between those two extremes.

For architecture re-launch planning, new scenarios extending the original scenarios can be identified for in-depth analysis and architectural refinement. The team uses the estimated number of new scenarios and the task-hour data from the previous cycle to develop a plan that includes a realistic schedule, while working to keep better track of reworked elements.

3.6 Launching the Developer Team

Prior to the implementation launch, some effort is needed to transition the existing architecture artifacts to the developer team for use in planning. An ARID-style peer review serves this purpose. ARID is a use case/scenario-based, stakeholder-centric review of a portion of an architecture, typically a software-invocable service. The ARID facilitator guides the developers in elaborating and applying scenarios to understand whether the architecture design is sufficient for the developers of the software that will use it.

First, a member of the architecture team gives an architecture presentation and traces scenarios through the architecture to illustrate key features. Developers read the documentation from the

point of view of the groups they are assigned to and the modules they will be responsible for developing.

Second, developers meet in their assigned groups to analyze a scenario with respect to their assigned modules. Members of the architecture team join the development groups to observe how the documentation is being used and to answer questions if the developers get stuck. Developers refer to the documentation and sketch a design that would fulfill the scenario as pseudo code, sequence diagrams, or language-specific interfaces. They can communicate with other teams as needed to negotiate interfaces. They are directed to try to use the documentation and to ask a member of the architecture team for help if they get stalled. In addition to answering questions, members of architecture team act as observers and write down all questions, interventions, and communications.

After the allotted time, the groups reconvene and present results to one another. The architects check to see that the pieces fit into a global solution that satisfies the scenario. The issues that surface are reviewed and feedback is given to the architecture team for improving the architecture.

Now that the architecture is well understood, it can serve as the blueprint for what parts are to be estimated during the implementation launch. The core of the launch consists of Meeting 3, Meeting 4, and Meeting 6, as successive estimations are made for the architectural modules. The participation of the team leader is key to providing critical information as to how each module is targeted for implementation over the next development cycle. The team estimates the entire remaining development and refines the estimates in detail for the next cycle. If the information is uncertain, the team iterates through the parts of Meeting 4 and Meeting 6 that deal with the next-cycle team and individual plans.

4 Pilot Application Experience of the Combined Approach

This section relates the experience of using TSP and ACE together in a project at Bursatec.

4.1 Project Summary (to Date)

In early 2009 Bursatec, the IT development organization of the Bolsa Mexicana de Valores (BMV)—the Mexican Stock Exchange—began planning a project to replace its electronic stock trading engine, which despite long and satisfactory service was beginning to show the pressure of rapidly expanding trading activity (daily average transaction volume more than tripled in 2009) and of being implemented on legacy and increasingly more expensive hardware.

The project had significant objectives and challenges from the start. The overall objective was simply stated: Implement a world-class trading system. A significant and expensive upgrade to the existing system had reduced transaction times, but the organization saw that additional improvements were needed to remain competitive with modern systems in the U.S., Europe, and Asia that have a lower processing latency. In addition to delivering transaction speed, the system must work flawlessly and unceasingly through the trading day. If that weren't enough, the stock exchange wanted to combine stock market trading with derivative market trading on the same platform to reduce operating costs and to provide a single high-throughput, low-latency, high-confidence interface to the outside financial world, increasing the overall availability of BMV and therefore Mexican companies to foreign capital.

The project challenges, though perhaps less quantifiable than the objectives, were no less significant. For one, the few remaining experienced developers from the existing system had either moved into management or possessed technical skills out of date with modern development technologies; the other developers available internally, while competent, were relatively inexperienced. For another, Bursatec wanted to adjust its management mechanisms and technical processes to cope with this project. For still another, there were significant voices within the organization in favor of outsourcing the development or even purchasing an existing trading engine.

Bursatec produced a plan and successfully made the case to its management to put the capabilities in place to execute the project internally. It brought in world-class management consultants to help create a project management office and contracted with the SEI for technical help in implementing ACE practices using TSP. Even though these two technologies have been highly successful on their own, they have no track record of being used together. Bursatec management recognized that it needed capabilities of both technologies in order to be successful, and there would be no second chance for this project. Bursatec had to get it right the first time.

The SEI began by focusing the organization on the expression of its business objectives in terms of quality attribute scenarios in a QAW and ensuring that the quality attributes were understood in the proper business context in a related Business Thread Workshop (BTW). The scenarios developed formally captured the non-functional requirements for high performance (transaction time), high availability (throughout the trading day), scalability (to allow for future growth), rapid modifiability (to allow for changes in business rules), and testability (to confirm proper function prior to deployment) that would be crucial to the project's success. Achievement of these quality

attributes would then be the driving factors for the activity of a small architecture team. In turn, this team would drive the entire development and testing effort.

In parallel with QAW/BTW preparations, managers and developers were trained in role-appropriate aspects of the Personal Software Process (PSP, mainly for developers but also a few key, technically-capable managers) and the TSP. The formal TSP launch of the architecture phase of the project was held the week following the QAW/BTW.

At the launch, an experienced TSP coach helped a talented but inexperienced architecture team bring together the various technical and business threads into an executable development plan. The project followed an iterative and incremental approach to software development. The quality attributes were used in the context of the ADD method to produce a software architecture that meets both functional and non-functional requirements using, in this instance, readily available commodity hardware and software. The design was documented with the Views and Beyond approach using an UML-based tool. In parallel, an implementation team was launched to develop an automated test framework and to evaluate a message communication bus that would provide the necessary backbone of the system.

The initial quality attribute scenarios, five in all, were used in combination with the ADD method to structure an iteration of 3 two-week bursts of activity for the architecture team. This pattern was based on an iterative development plan featuring 3 six-week iterations, with two-week "mini-iterations" built in to focus the team on short-term technical objectives. The team would focus on two scenarios over two weeks, formulating and elaborating architecture views to address current scenarios while remaining consistent with previous ones and capturing the results in preparation for a visit from an experienced architecture coach. The coach critiqued the current product using ATAM-based analysis techniques, prodded the team to capture critical decisions while guiding them to properly capture critical architectural information, and then helped to elaborate the details of the plan for the following two weeks. The third such iteration elaborated architectural elements for the last of the five major scenarios and also elaborated various sub-scenarios rooted in the initial five.

These sub-scenarios were then used in a planned "re-planning" session, a common activity for a TSP team, to revise the plan for the next six-week iteration that would result in a *Version 1* of the architecture—something fit for initial use by the implementation team and eventually a formal architecture evaluation. This iteration was spread over end-of-year holidays, so instead of the earlier two-week intervals between visits by the architecture coach, more than a month passed. When the coach arrived early in the new year for his next session, he recognized a difference in his team. They were no longer just good young developers learning architectural methods on the job, they were architects.

The Version 1 architecture was used to launch the implementation phase of the project. ARID-based peer reviews served to transition the architecture to the developer team. Under SEI guidance, elements of the ARID method were used to put the architecture documentation in the hands of the developers, ensure that the documents were fit for development use, and provide feedback to the architecture team. This initial use of the architecture documentation by the people who would develop this world-class trading engine was followed immediately by a launch of the initial implementation phase of the project. This phase developed the basic data and communication infrastructure for using the commodity communications product. The architecture team meanwhile

launched its last cycle of activity principally as architects, as this group would essentially merge into the implementation team in the next cycle. The principle objectives of this phase were to produce the *Version 2* architecture ready for review by a formal evaluation using the ATAM.

The evaluation using the ATAM showed that that architecture design was remarkably successful. The ATAM is focused on identifying and surfacing architectural risks, yet this evaluation identified fewer risks than expected for a project of this size and scope. The architecture documentation, often cited as a shortcoming in ATAMs, was instead identified as a particular strength, most likely because the architecture coach had put the team through "boot camp" for the previous few months, carefully setting the scope and incrementally documenting the architecture during each visit. This method, while particularly intensive in the use of a scarce architecture coach resource, was very effective in producing a complete and thorough architecture document, in addition to a team of competent architects.

The next launch marked the merging of the architecture and implementation teams into a single team with a common launch. The main objective of this phase was highly visible and likely to be indicative of ultimate success—namely, the implementation of the main trading mechanism on top of the data and communications infrastructure. This cycle was finished on time with a minimum of extra effort (defined as very late nights and weekends). Early testing results indicate excellent performance on the examples.

As of this writing, the team has completed planning for the implementation of the majority of remaining functions as well as the other critical quality attribute, high availability. Also, more performance tuning is in the plan, based on recommendations from an outside expert and in response to the implementation of a new requirement that potentially touches every part of the processing cycle for trades.

4.2 Important Lessons Learned (So Far)

TSP and ACE are different disciplines founded on core principles (see Table 9). TSP is a self-directed management and measurement process. ACE is the discipline of using architecture as the focal point for performing ongoing analyses to gain increasing levels of confidence that systems will support their missions.

Table 9: ACE and TSP Principles

ACE Principles	TSP Principles
A software architecture should be defined in terms of elements that are coarse enough for human intellectual control and specific enough for meaningful reasoning.	High-performance teams plan, manage, and own their commitments.
Business goals determine quality attribute requirements; quality attribute requirements guide the design and analysis of software architectures.	A disciplined planning and measurement framework with personal reviews and team inspections helps engineers ensure quality software products.
Architecture-centric activities drive the software system life cycle.	TSP provides a disciplined framework for measuring and managing any structured intellectual activity.

In combination, TSP and ACE principles are supported by common techniques that emphasize business and quality goals, engineering excellence, defined processes and process discipline, and teamwork. They allow TSP and ACE to work well together.

TSP guides the team in putting the planning and measurement framework in place. Without ACE practices, the team would have likely proceeded directly from the conceptual design to detailed design and coding, without the benefit of the software architecture to reason about design quality early in the life cycle. The team would not have had the benefit of using the quality attributes early to structure the design and using the design to drive the downstream life-cycle activities of implementation, test, integration, and validation.

ACE practices guide the team in putting the architecture in place. Without TSP, the lead architect would likely have had to shoulder more responsibility for developing the architecture and would not have had the benefit of the TSP coach and framework to help guide and train the rest of the architecture team. The team would not have had the benefit of using the framework to show progress to management and to show how the architecture artifacts relate to the broader software development plan. The following lessons have been learned so far:

- TSP and ACE played complementary roles. TSP and ACE provided a disciplined approach across the life cycle for developing software that meets its business goals and quality attribute requirements. TSP brought discipline and measurement to a set of robust architectural techniques that are focused on meeting business goals and quality requirements. ACE provided clear direction for architecture-related activities early in a project life cycle, whereas TSP has traditionally focused on implementation later in the life cycle.
- An architecture coach complemented the TSP coach. Just as TSP has a role of a team coach to help the team get started and to participate in weekly meetings, there was a need for a similar role of architecture coach. The architecture coach helped the team get started during the launch and participated in the biweekly meetings. Coaching consisted of two parts—asking questions to review progress and pushing the team in the right direction to tackle the next problem.
- The combined approach helped guide and train the junior architects. Initially, the roles of team leader and lead architect were assigned to the single individual who had the needed skills and experience. Over the course of the architecture design, the three architects-in-training developed their architecture skills and assumed more responsibility. At the architecture team re-launch, the architecture lead maintained team leader responsibility and took the role of architecture coach, with the junior architects assuming the lead architect role.
- The combined approach facilitated an iterative and incremental approach to design and implementation. Starting the development cycles early forced the architecture team to quickly produce a description of architecture elements that would be understandable by the developers. This kept the architects focused on the needs of developers, one of their important stakeholder groups. This interaction broke down barriers between the architects and the developers, enabled early feedback from the developers and ensured that architecture documentation was developed naturally and not as a separate task.
- Architecture embedded in the TSP framework provided management early visibility into the team progress. TSP's planning and measurement framework provided a disciplined approach to roll out architecture practices, help engineers ensure quality software products, and provide senior management and the program office with visibility into architecture progress and quality during early stages of the project.

- An architecture-centric approach provided early measures of quality. The peer reviews that were integrated into the design process by the architecture coach helped keep the design on track in meeting its quality attribute goals and resulted in finding more non-risks than risks during the evaluation using the ATAM.

5 Summary

TSP and ACE are different disciplines. TSP is a self-directed management and measurement process, while ACE is a collection of technical development practices. However, shared emphasis on business and quality goals, engineering excellence, defined processes and process discipline, and teamwork allow TSP and ACE to work well together.

Combining ACE and TSP provides an iterative approach for delivering high quality systems on time and within budget. TSP provides the infrastructure in estimation, planning, measurement, and project management. ACE provides the means for designing, evaluating, and implementing a system so that it will satisfy its business and quality goals. The combined approach offers help to organizations that have a need to set an architecture/developer team in motion using mature, disciplined engineering practices that produce quality software quickly.

The approach has been piloted on a project at Bursatec where teams continue to follow this disciplined process of planning, tracking, and gathering data as development continues. They have used initial data to adjust the plan and meet interim commitments and milestones. The customer is pleased with the integration of ACE and TSP methods and will similarly launch additional project teams this year.

Appendix Recommended Training

One way to gain knowledge of ACE practices and TSP is through training from the SEI or its licensed partners. Different pathways through the courses are available depending on the role of the individual with respect to the project.

Software architecture training begins with courses for the lead architect, technical managers, and engineers on a path to improve their architecting skills.

1. Course: *Software Architecture: Principles and Practices*. Required course introduces the essential concepts of architecture.
2. Course: *Software Architecture Design and Analysis*. Required instruction that explores design and analysis in-depth through the application of the three methods that are used by the architecture team, the QAW, the ADD method, and the ATAM.

TSP training begins with courses for executives, middle and line managers, and members of the development teams involved in the initial pilot projects. Three classes are delivered in preparation for the implementation of TSP:

1. Course: *TSP/PSP Executive Overview and Planning Session*. Required session to introduce senior management to the key concepts, benefits and requirements of the PSP/TSP and prepare senior management to plan and implement TSP.
2. Course: *Leading TSP Development Teams*. Required instruction for all managers who directly manage software development: software project managers, software team managers, and supervisors. This course introduces the quantitative TSP project management and quality management concepts that managers use to build high-performance TSP teams.
3. Course: *PSP Fundamentals*. Required instruction for all engineers who will be on a software development team using TSP. This course introduces both PSP and TSP methods that development team members need to apply TSP.

The SEI recommends that organizations using architecture and TSP develop their own coaches. Coach development begins with advanced courses and includes an observation component to ensure candidates have the necessary qualifications.

Architecture coach development follows the path:

1. Certificate: *Software Architecture Professional*. Two additional courses (beyond those previously mentioned for software architecture training) in the software architecture curriculum provide the architect with exposure to needed skills, *Documenting Software Architecture* and *Software Product Lines*.
2. Certification: *ATAM Leader Certification*. Provides a training path for someone in the role of architecture coach. Three additional courses (beyond those previously mentioned for software architecture training) provide the architect with exposure to needed skills, *Documenting Software Architecture, ATAM Evaluator Training* and *ATAM Leader Training*. Candidate leaders who successfully complete the ATAM Leader training must also complete the *ATAM Leader Observation* in order to become an SEI-certified ATAM Leader. The ability to ana-

lyze architecture is one of the two essentials skills of a coach. The other is the ability to design, for which there is no classroom substitute for substantive experience in the field.

TSP coach development follows the path:

1. Certification: *PSP Developer*. Prior to entering the coach training program, the candidate coach must become a PSP certified developer by taking either PSP Fundamentals and PSP Advanced or PSP I and PSP II followed by the PSP Certification examination.
2. It is strongly recommended the candidate TSP coach also take the course *Leading a Development Team* and participate on a TSP team as either a developer or team lead.
3. Course: *TSP Coach Training*. Required instruction that prepares the participant to coach teams using TSP and to become a Provisional TSP Coach.
4. Certification: *TSP Coach*. Candidate coaches who successfully complete the TSP Coach training must then coach a team through a launch, checkpoint, and post mortem under the guidance of an SEI-certified Mentor Coach. Finally, the Provisional Coach must successfully complete the TSP Coach Certification examination in order to become an SEI-certified TSP Coach.

References

URLs are valid as of the publication date of this document.

[Barbacci 2003]
Barbacci, M. R., Ellison, R., Lattanze, A. J., Stafford, J. A., Weinstock, C. B., & Wood, W. G. *Quality Attribute Workshops (QAWs), Third Edition* (CMU/SEI-2003-TR-016, ADA418428). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. http://www.sei.cmu.edu/library/abstracts/reports/03tr016.cfm

[Bass 2003]
Bass, L., Clements, P., & Kazman R. *Software Architecture in Practice, Second Edition*. Boston, MA: Addison-Wesley, 2003.

[Bass 2006]
Bass, L., Nord, R., Wood, W., & Zubrow, D. *Risk Themes Discovered Through Architecture Evaluations* (CMU/SEI-2006-TR-012). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2006. http://www.sei.cmu.edu/library/abstracts/reports/06tr012.cfm

[Cervantes 2010]
Cervantes, H., Martinez Aceves, I., Castillo, J., & Montes de Oca, C. "Introducing Software Architecture Development Methods into a TSP-based Development Company." *SEI Architecture Technology User Network (SATURN) Conference, 2010*. Minneapolis, MN, May 17-21, 2010. http://www.sei.cmu.edu/library/abstracts/presentations/cervantes-saturn2010.cfm.

[Clements 2002]
Clements, P., Kazman, R., & Klein, M. *Evaluating Software Architectures: Methods and Case Studies*. Boston, MA: Addison-Wesley, 2002.

[Clements 2003]
Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., & Stafford, J. *Documenting Software Architectures: Views and Beyond*. Boston, MA: Addison-Wesley, 2002.

[Edmondson 2007]
Edmondson, J. S., Lee, E., & Kille, C. G. "A Light-weight Architecture Trade Off Process Based on ATAM." *SEI Architecture Technology User Network (SATURN) Conference, 2007*. Pittsburgh, PA: May 14-16, 2007. http://www.sei.cmu.edu/library/abstracts/presentations/ATO-Lite-for-SATURN-2007-2.cfm

[Forstrom 2008]
Forstrom, H. "Inexpensive ATAM-Peer Review Detects and Fixes Architecture Problems Early." *SEI Architecture Technology User Network (SATURN) Conference, 2008*. Pittsburgh, PA: April 30-May 1, 2008. http://www.sei.cmu.edu/library/abstracts/presentations/ATAM-peer-review-SATURN-2008.cfm

[Humphrey 2002]
Humphrey, W. S. *Winning with Software: An Executive Strategy*. Boston, MA: Addison-Wesley, 2002.

[Humphrey 2005]
Humphrey, W. S. *PSP: A Self-Improvement Process for Software Engineers*. Boston, MA: Addison-Wesley, 2005.

[Humphrey 2006a]
Humphrey, W. S. *TSP: Leading a Development Team*. Boston, MA: Addison-Wesley, 2006.

[Humphrey 2006b]
Humphrey, W. S. *TSP: Coaching Development Teams*. Boston, MA: Addison-Wesley, 2006.

[Kazman 2004]
Kazman, R., Kruchten, P., Nord, R. L., & Tomayko, J. E. *Integrating Software-Architecture-Centric Methods into the Rational Unified Process* (CMU/SEI-2004-TR-011). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004. http://www.sei.cmu.edu/ library/abstracts/reports/04tr011.cfm

[Nichols 2009]
Nichols, W.R. & Salazar, R. *Deploying TSP in a National Scale: An Experience Report from Pilot Projects in Mexico* (CMU/SEI-2009-TR-011). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2009. http://www.sei.cmu.edu/library/abstracts/reports/09tr011.cfm

[Nord 2004]
Nord, R. L., Tomayko, J. E., & Wojcik, R. Integrating Software-Architecture-Centric Methods into Extreme Programming (XP) (CMU/SEI-2004-TN-036). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004. http://www.sei.cmu.edu/library/abstracts/reports/04tn036.cfm

[Wojcik 2006]
Wojcik, R., Bachmann, F., Bass, L., Clements, P., Merson, P., Nord, R., & Wood, W. *Attribute-Driven Design (ADD), Version 2.0* (CMU/SEI-2006-TR-023). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2006. http://www.sei.cmu.edu/library/abstracts/reports/06tr023.cfm

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	December 2010	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Combining Architecture-Centric Engineering with the Team Software Process	FA8721-05-C-0003

6. AUTHOR(S)

Robert L. Nord, James McHale, Felix Bachmann

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2010-TR-031

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	ESC-TR-2010-031

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

This report contains a description of an architecture-centric life-cycle model that uses the Carnegie Mellon Software Engineering Institute's architecture-centric engineering (ACE) methods embedded in a Team Software Process (TSP) framework and our experience in piloting the approach in an actual development effort. Combining ACE and TSP provides an iterative approach for delivering quality systems on time and within budget. TSP provides the infrastructure in estimation, planning, measurement, and project management. ACE provides the means for designing, evaluating, and implementing a system so that it will satisfy its business and quality goals. Bringing these approaches together results in something that is much more than the sum of the parts. The combined approach offers help to organizations to set an architecture/developer team in motion using mature, disciplined engineering practices that produce quality software quickly.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Architecture-centric engineering, Team Software Process, ACE, TSP	48

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL